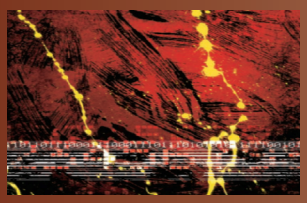
Self-Managing Software

Michael G. Hinchey
NASA Software Engineering Laboratory

Roy Sterritt
University of Ulster



The properties of an autonomic or self-managing system provide the basis for future selfware development.

Editor's note: *Contributions to this new bimonthly column will examine leading-edge software development trends, including software development tools and methodologies, languages, formal methods, and software engineering. Readers interested in submitting column-length essays (2,000-2,500 words) should contact Michael G. Hinchey at michael.g.hinchey@nasa.gov.*

Software has become pervasive. It helps entertain us when embedded in iPods and MP3 players. In GPS navigation systems, it helps us get to our desired location. It's present in our home appliances and cell phones. It's used in defense systems, space exploration, and hospitals. Software constantly appears in more devices, with public expectations of greater functionality, reliability, and continued operation.

Despite this success and expansion into daily life, there have, of course, been a number of software-related disasters and near-disasters. Software failures have resulted in giving cancer patients excessive (and lethal) doses of radiation, loss of aircraft and spacecraft, and disclosures of private financial information.

We continue to push software to the limits, in many cases using it where failure would be catastrophic, and where many organizations are spending as much as 33 to 50 percent of the total cost of ownership of their computing and communication systems to avoid software failure.

Many practitioners believe that self-managing software can potentially ensure safer, more reliable, and cost-effective computer systems. Creating software systems that are self-directed, self-governing, and self-adapting has been the focus of development in autonomic computing, autonomic communications, pervasive computing, organic computing, and adaptive computing.

AUTONOMIC SYSTEMS

In 2001, IBM launched its perspective on the state of IT (www.research.ibm.com/autonomic/manifesto), which focuses on solving the ever-increasing complexity and total cost of ownership of today's systems through the development of self-managing systems, a major component of which is self-managing software, inspired by the human body's *autonomic nervous system*. The ANS is that part of the nervous system that manages body functions nonconsciously such as blood circulation, intestinal activity, and hormonal secretion and production and thus lets us get on with our conscious daily lives.

The general properties of an autonomic, or self-managing, system can be summarized by four objectives—and four attributes. Essentially, the objectives represent broad system requirements, while the attributes identify basic implementation mechanisms.

An autonomic system's objectives are

- *Self-configuration.* The system must be able to readjust itself automatically, either to support a change in circumstances or to assist in meeting other system objectives.
- *Self-healing.* In reactive mode, the system must effectively recover when a fault occurs, identify the fault, and, when possible, repair it. In proactive mode, the system monitors vital signs to predict and avoid health problems, or reaching undesirable levels.
- *Self-optimization.* The system can measure its current performance against the known optimum and has defined policies for attempting improvements. It can also react to the user's policy changes within the system.
- *Self-protection.* The system must defend itself from accidental or malicious external attacks, which requires an awareness of potential threats and the means to manage them.

To achieve these self-managing objectives, a system must be

- *self-aware*—aware of its internal state;
- *self-situated*—aware of current external operating conditions;
- *self-monitoring*—able to detect changing circumstances; and
- *self-adjusting*—able to adapt accordingly.

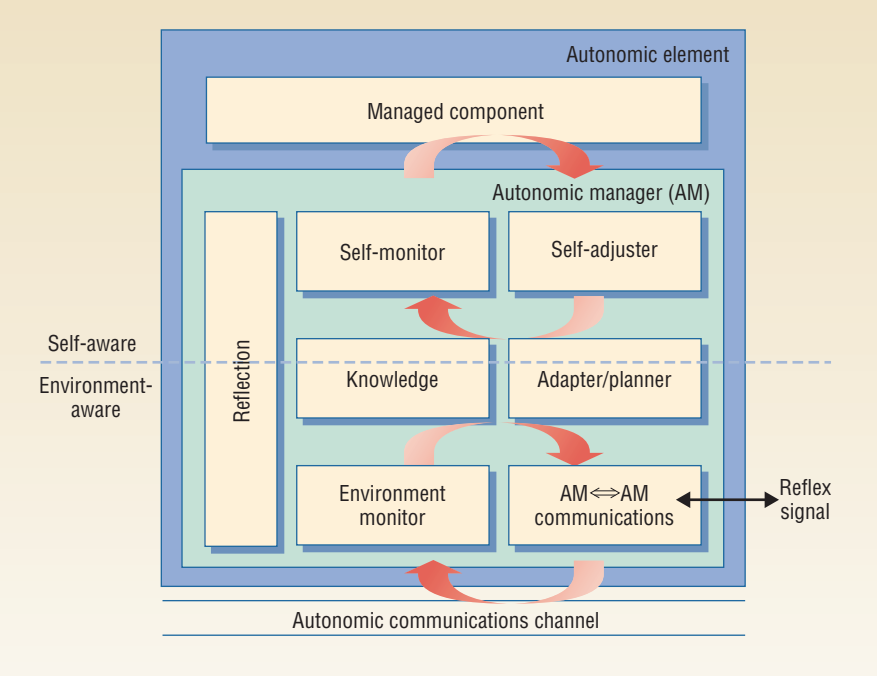


Figure 1. The control loops in an autonomic element. The autonomic manager to autonomic manager communications include a reflex signal.

Thus, a system must be aware of its available resources and components, their ideal performance characteristics, and current status. It must also be aware of interconnection with other systems, as well as rules and policies for adjusting as required. A system's ability to operate in a heterogeneous environment requires relying on open standards to communicate with other systems.

These mechanisms do not exist independently. For example, to successfully survive an attack, the system must exhibit self-healing abilities, with a mix of self-configuration and self-optimization. This not only ensures the system's dependability and continued operation but also increases self-protection from similar future attacks. Self-managing mechanisms must also ensure minimal disruption to users.

CONTROL LOOP

Sensors and effectors are central to the architecture of any autonomic system. An autonomic manager creates a control loop by first monitoring behavior through sensors and then comparing it to historical and current data, rules, and beliefs. The autonomic manager then plans the action, if necessary, and executes it through effectors. The closed feedback control loop provides the basic backbone for each system component. Figure 1 illustrates the two conceptual control loops, self-awareness (managed component) and self-situation (environment), in an autonomic element.

IBM represents this self-monitoring/self-adjusting function as the monitor, analyze, plan, and execute (MAPE) control loop. The monitor and analyze aspects process information from the sensors to provide both self-awareness and environmental awareness. The plan and execute aspects control the self-management behavior that the effectors will execute.

The MAPE components use correlations, rules, beliefs, expectations, histories, and other information either directly known to the autonomic element or otherwise available through the knowledge repository within the autonomic manager.

Reflex signals

To ensure its robustness, the autonomic environment requires the autonomic elements and, in particular, autonomic managers, to communicate with each other regarding the various self-* activities. The communication between autonomic managers in Figure 1 also includes a reflex signal, which a pulse monitor—with the capability to encode system health and urgency signals as a pulse—can facilitate.

The pulse monitor, an extension of the embedded system's heartbeat monitor, safeguards vital processes through the emission of a regular "I am alive" signal to other processes. Just as a human heart has a double beat, the pulse monitor has an encoded double beat—a self health/urgency measure and an environment health/urgency measure—that corresponds with the autonomic element's self- and environmental-awareness control loops.

Together with the standard event messages on the autonomic communications channel, this information provides dynamics within autonomic responses and multiple control loops, such as reflex reactions among the autonomic managers.

This reflex component safeguards the autonomic element by communicating its health to another autonomic element. The component can also communicate environmental health information. For example, instead of each individual PC in a LAN—all equipped with an autonomic manager—monitoring the same environment, some can take on the role of alerting others through a change in pulse to indicate changing circumstances.

Minimizing sent data—essentially transmitting only a signal—is an important aspect of pulse monitoring to facilitate a reflex reaction. In the absence of bandwidth concerns, the autonomic manager can send more detailed information, but the additional information must be in a form that will not compromise the reflex reaction. For example, the autonomic manager could send detailed health telemetry in a form that does not incur processing delays and can be acted on immediately.

SELFWARE

The initial set of self-* properties, commonly known as self-CHOP (con-

figuring, healing, optimizing, and protecting) represents the general goal of the Autonomic Computing Initiative. These properties are neither mutually exclusive nor definitive. Practitioners have began to propose and develop more self-* properties, which has led to coining the term selfware. We currently seek inspiration for new approaches to selfware from existing biological mechanisms. An example of one new self-* property is self-destruct.

Safety mechanisms

To provide an intrinsic safety mechanism, for example, against undesirable emergent selfware behavior or in response to security concerns, researchers are investigating the need for software agents or component self-destruction. To illustrate, skin cells from a cut are often displaced into muscle tissue. If these cells survive and divide, a tumor can result. The body's solution to this is cell self-destruction, referred to as *apoptosis*. The concept originated from Greek word to "fall off," referring to leaves falling from trees in the autumn. In this context, apoptosis is the death of cells in the midst of a living structure.

Self-destruction

Some researchers believe that cells know when to self-destruct because they are programmed to do so. This intrinsic biological self-destruction property—also referred to as death-by-default—is delayed through the continuous receipt of biochemical reprieve signals. Mounting evidence now suggests that some forms of cancer are the result of cells not dying fast enough, rather than multiplying in an out-of-control manner, as commonly thought.

When a cell divides, it receives a simultaneous order to self-destruct, and it will do so in the absence of a reprieve signal. Self-protection is the reason for this. Cells must divide for the body to survive, but this is a dangerous time because if just one of the billions of cells locks into division, a tumor results.

Likewise for computing systems, while a self-managing agent's binary image—which can contain, for example, passwords, monetary certificates, or confidential information—might be protected by encryption, it must decrypt to execute, providing a window of vulnerability. These self-protection needs are similar to our own bodies' during cell division, which is protected by apoptosis.

Self-managing software, viewed from the perspective of autonomic computing or other selfware initiatives, offers a holistic vision for software's development and evolution, bringing new levels of automation, autonomy, and dependability to systems, while simultaneously hiding their complexity and reducing costs. We envisage greater interest in, and uptake of, self-managing principles in future software development, as demand increases for systems to exhibit more autonomic properties. The IEEE Computer Society recently established a Task Force on Autonomous and Autonomic Systems (www.computer.org/portal/pages/ieeecs/Communities/tab/tclist/TFAAS.html). Interested readers can sign up to join the task force at www.computer.org/TCSignup.

Michael G. Hinchey is director of the NASA Software Engineering Laboratory at NASA Goddard Space Flight Center and an affiliate professor at Loyola College in Maryland. Contact him at michael.g.hinchey@nasa.gov.

Roy Sterritt is an academic in the School of Computing and Mathematics at the University of Ulster, Northern Ireland, and a researcher in the Computer Science Research Institute and the Centre for Software Process Technologies. Contact him at r.sterritt@ulster.ac.uk.